Two new forms of the standard UPC collective functions offer higher productivity through more expressive interfaces

One-Sided Collectives

**Definition:** A *one-sided collective* is an operation initiated by one thread that accesses or changes memory in all threads’ partitions.

Standard UPC contains one-sided collective library operations, such as `upc_global_alloc()`. One-sided alternatives to the standard data movement and computational collectives are now being evaluated on several platforms.

**Data Movement Collectives:**

```c
void mupc_broadcast(shared void *src, size_t n)
```

The `n` bytes at `src` are copied to corresponding areas of shared memory on all threads. The broadcast is “in place”. `src` need not have affinity to the caller. The broadcast is complete at the beginning of the next synchronization phase.

**Computational Collectives:**

```c
TYPE mupc_reduce
```

A reduction with operation `op` is performed over the `n` corresponding elements of `src` of type `TYPE` on all threads. The result is available to the calling thread upon return.

Usage comparison:

Find the min and max of shared array `A`:

```c
#define N 10000
shared [N] double A[N*THREADS];
shared [] double MinMax0[2]; // targets of reduction
shared [2] double MinMax[2*THREADS]; // targets of broadcast

upc_all_reduceD( &MinMax0[0], A, UPC_MIN, N*THREADS, N, NULL, UPC_IN_ALLSYNC | UPC_OUT_MYSYNC );
upc_all_reduceD( &MinMax0[1], A, UPC_MAX, N*THREADS, N, NULL, UPC_IN_MYSYNC | UPC_OUT_MYSYNC );
upc_all_broadcast( MinMax, MinMax0, 2*sizeof(double), UPC_IN_MYSYNC | UPC_OUT_ALLSYNC );
```

Value-Based Collectives

**Definition:** A *value-based collective* is a wrapper that provides a “one-liner” for scalar collective operations.

Value-based collectives are implemented as a generic header file that works with any UPC-1.2 compliant compiler.

**Data Movement Collectives:**

```c
TYPE bupc_allv_broadcast(TYPE, TYPE value, int rootthread)
```

```c
TYPE *bupc_allv_gather_all(TYPE, TYPE *destarray)
```

`TYPE` may be scalar or aggregate (struct or union) type. Array parameters are pointer-to-local for the calling thread. Also have variants for scatter, gather and permute.

**Computational Collectives:**

```c
TYPE bupc_allv_reduce_all(TYPE, TYPE value, upc_op_t op)
```

`TYPE` must be scalar. A reduction with operation `op` is performed over the values provided by all threads. The result is returned to all threads. Also have variants for reduce and prefix_reduce.

Usage comparison:

Find the min and max of shared array `A`:

```c
#define N 10000
shared [] double *lA = (shared []) double *)&A[N*MYTHREAD];
double localmin = lA[0], localmax = lA[0];
// compute localmin & localmax over my data...
for (int i=1; i < N; i++) {
    localmin = MIN(localmin, lA[i]);
    localmax = MAX(localmax, lA[i]);
}
// perform the collective reduction
double min = bupc_allv_reduce_all(double, localmin, UPC_MIN);
double max = bupc_allv_reduce_all(double, localmax, UPC_MAX);
```